

# Speculative Deforestation

William Sonnex

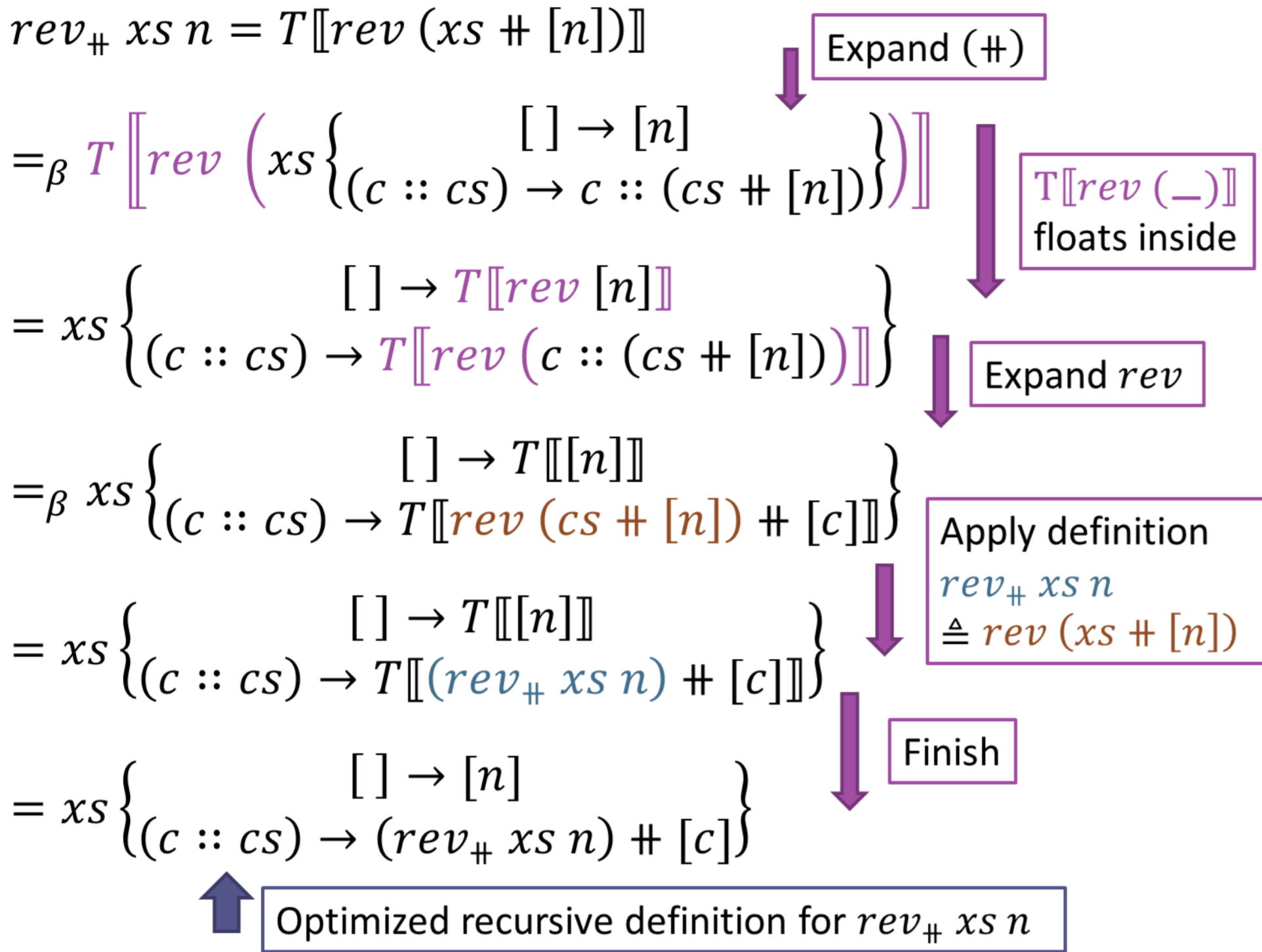


## 2. Deforesting $rev\ xs\ \# [n]$

Fix a new  $rev_{\#} : [a] \rightarrow a \rightarrow [a]$ , s.t.

$$rev_{\#} xs\ n \triangleq rev\ xs\ \# [n]$$

Deforest the definition of  $rev_2$ :

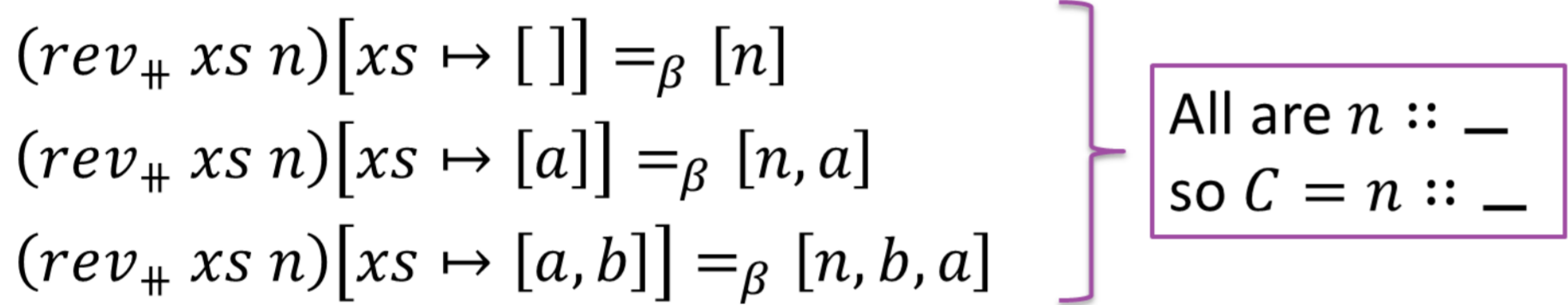


## 3. Factoring $rev_{\#} xs\ n$ into $n :: (rev\ xs)$

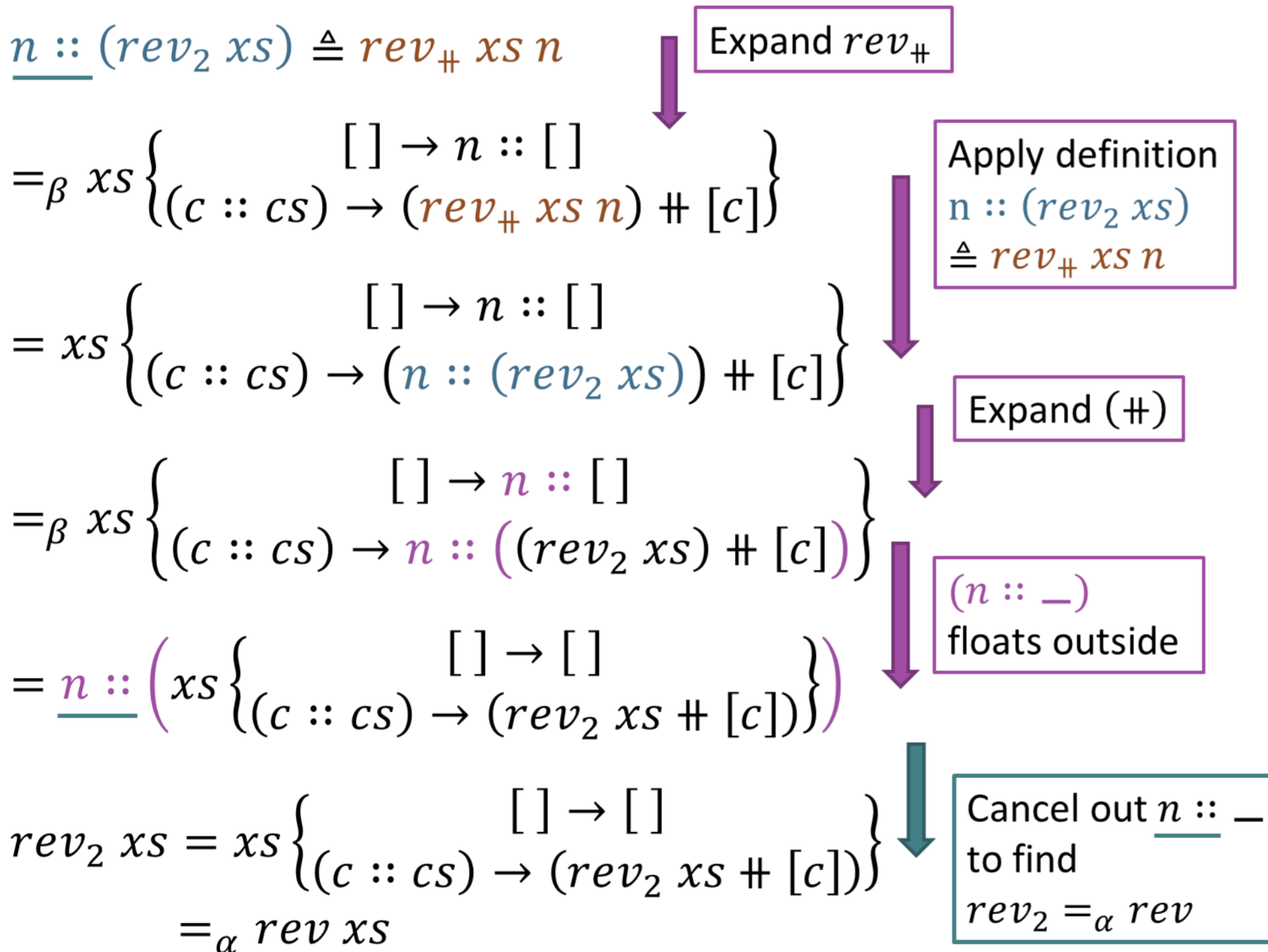
Deforestation found  $rev\ xs\ \# [n] \rightsquigarrow rev_{\#} xs\ n$

Constant context factoring finds  $rev_{\#} xs\ n \rightsquigarrow n :: (rev\ xs)$

First, speculate the constant context  $C$  using a dynamic approach, enumerate inputs to  $rev_{\#} xs\ n$ :



Fix a new  $rev_2 : [a] \rightarrow [a]$  s.t.



## Function definitions

(List append)	$as\ \# bs = as\ \left\{ \begin{array}{l} [] \rightarrow bs \\ (c :: cs) \rightarrow c :: (cs\ \# bs) \end{array} \right\}$
(List reversal)	$rev\ ds = ds\ \left\{ \begin{array}{l} [] \rightarrow [] \\ (e :: es) \rightarrow (rev\ es)\ \# [e] \end{array} \right\}$
(Natural number addition)	$x + y = x\ \left\{ \begin{array}{l} 0 \rightarrow y \\ s(z) \rightarrow s(z + y) \end{array} \right\}$

## 1. Introduction

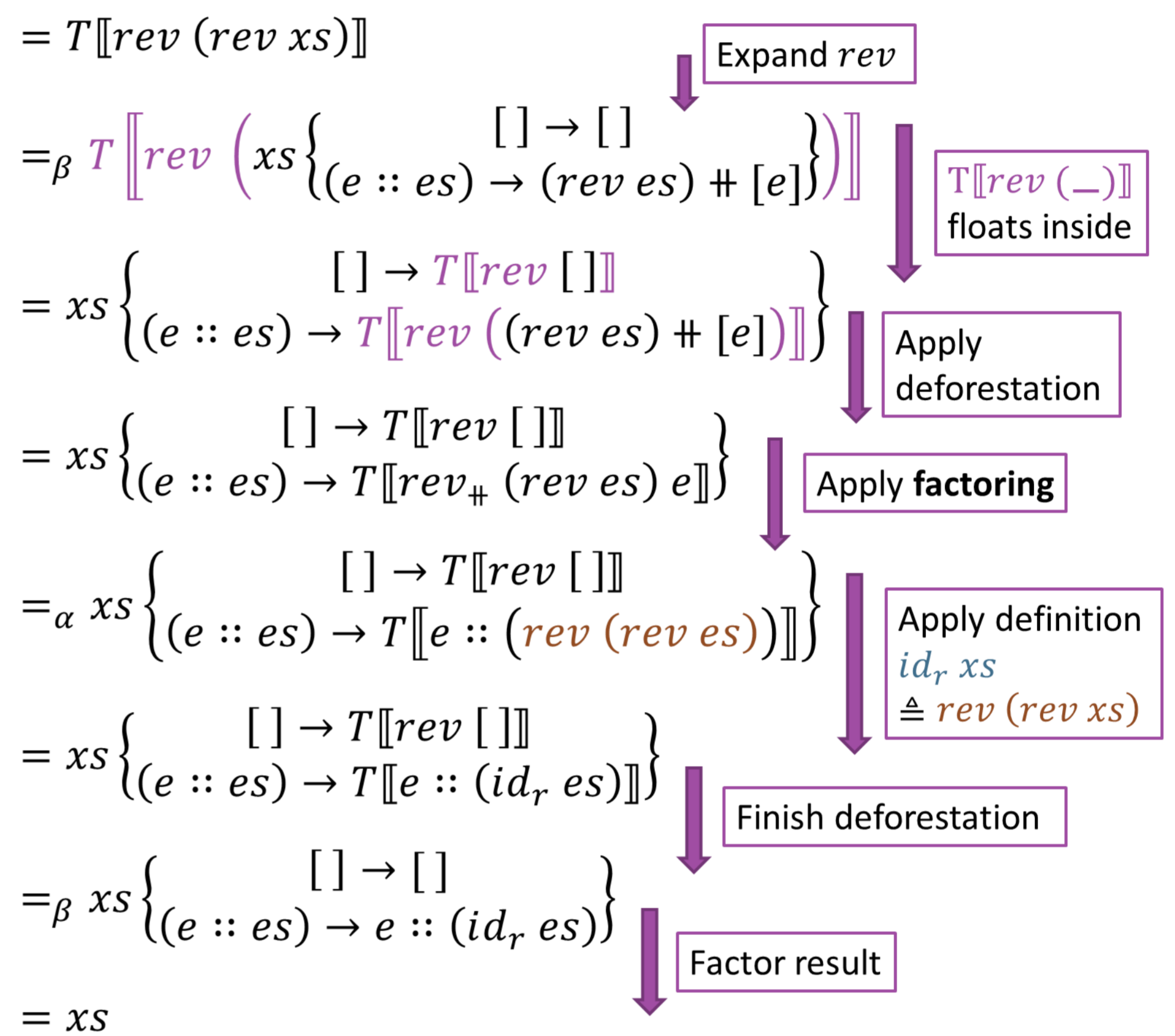
- Deforestation is a function simplification technique invented by Philip Wadler
- It optimises functional programs by removing intermediary results e.g.  $map\ g\ (map\ f\ xs) \rightsquigarrow map\ (g \circ f)\ xs$
- We have developed extensions to yield simpler results, not for runtime, but for program verification and ATP
- These extensions we have collectively named “factoring”, as they factor out a context from a recursive function, i.e.  $\mu h \rightsquigarrow f(\mu g)$ , factoring  $f$  out of  $\mu h$  to yield  $\mu g$
- In this poster we present only “constant context” factoring

## 4. Deforesting $rev\ (rev\ xs)$

With factoring we can now calculate:  $rev\ (rev\ xs) \rightsquigarrow xs$

Fix a new  $id_r : [a] \rightarrow a \rightarrow [a]$ , s.t.

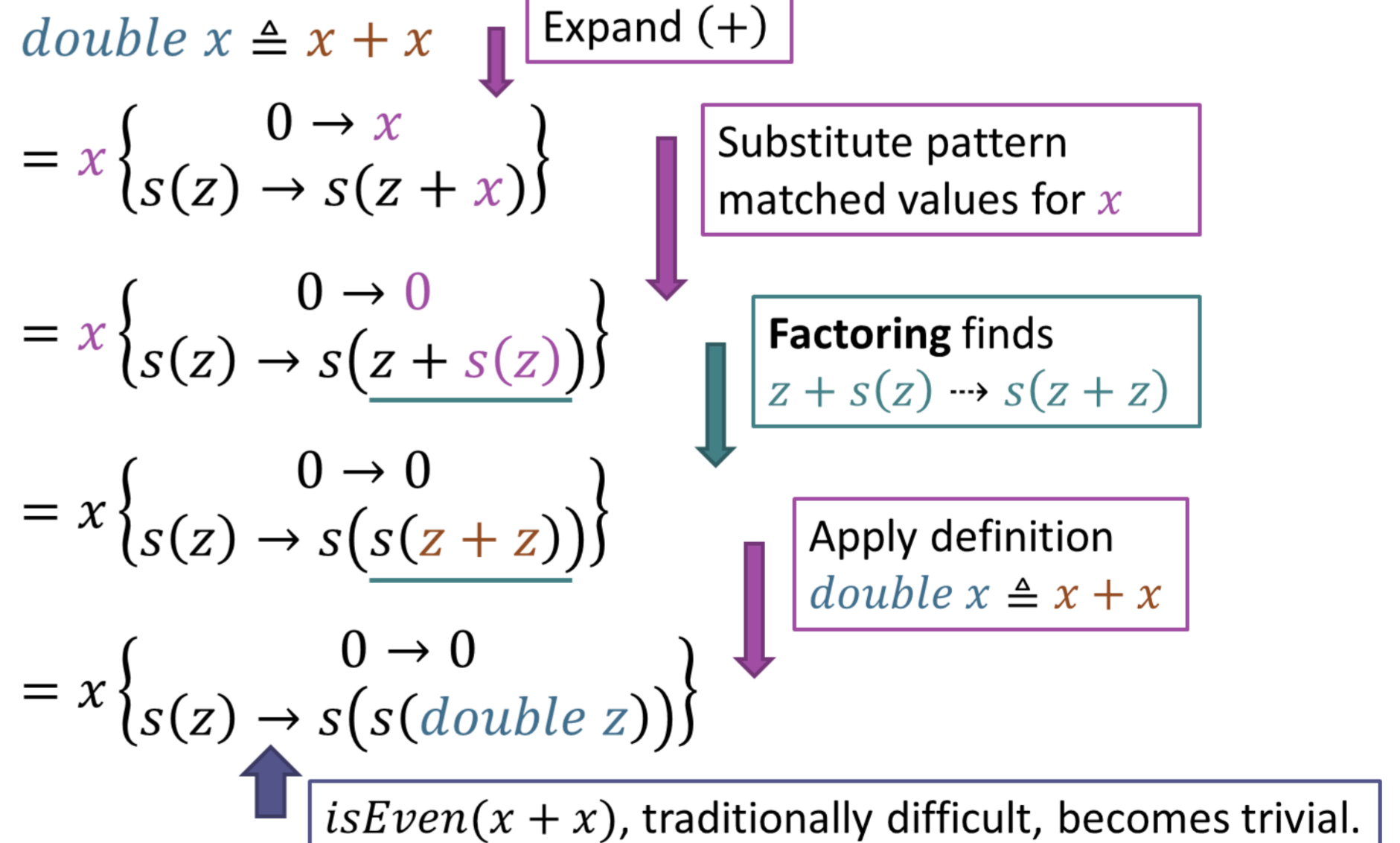
$$id_r\ xs \triangleq rev\ (rev\ xs)$$



## 5. Deforesting $x + x$

Factoring can also remove variable repetition, as in  $x + x$

Fix a new  $double\ x$  s.t.



## 6. Results

$length\ (rev\ xs) \rightsquigarrow length\ xs$	} Other results of just constant context factoring
$length\ xs\ \# xs \rightsquigarrow double\ (length\ xs)$	
$elem\ n\ (xs\ \# [m]) \rightsquigarrow n = m \vee elem\ n\ xs$	} Results of our full method
$count\ n\ (insertsort\ xs) \rightsquigarrow count\ n\ xs$	
$take\ (length\ xs)\ (rev\ xs) \rightsquigarrow rev\ xs$	
$treesort\ xs \rightsquigarrow insertsort\ xs$	